**Coupon system**

**Phase one**

***General information***

The coupon system enables companies to create coupons as part of commercial campaigns that they create.  
The system has listed customer and they can purchase the coupons.  
The system has 3 main users: Administrator, companies and customers.

***Assumptions and remarks:***

* The methods getCoupons (in CustomerDBDAO and CompanyDBDAO classes) were modified and a Coupon attribute and Company attribute respectively
* Configuration class: this class needs to be modified before running to program.

Emails need to be updated in order to support the delete coupon mailing process (explained late)

* The methods getCoupon, getCustomer and getCompany (that get id) were overloaded to get a string. Since the id in the database is a sequence and at the time of preforming the operations the id is unknown we must rely on the name. since the name is a unique we can address it as a key
* New methods were added to support the business logic implementation:
  + checkDbExistence: this method overloaded in all the DAOs and it checks id an instance with the same name (Customer, Company or Coupon) exists in the system.
  + isCopuonBelongsCompany: this method checks if the operation the company intending to preform will be done only on its coupons
  + getCouponsByType: this method overloaded in Customer and Company DAOs and return the coupons based on the Coupon type the customer or the company asks
  + getCouponsUpToPrice: this method overloaded in Customer and Company DAOs and return the coupons based on the price the customer or the company asks
  + getCouponsUpToDate: this method overloaded in Customer and Company DAOs and return the coupons based on the date the customer or the company asks
  + hasAlreadyPurchased: this method validates the constraint of customer can purchase a coupon only once
  + createCompanyCouponRecord: this method inserts a new record to Company\_Coupon table in cases a company creates a coupon
  + createCustomerCouponRecord: this method inserts a new record to Customer\_Coupon table in cases a customer purchases a coupon
* Emailing process: this process sends an email every 24 hours with a list of all the deleted-expired coupons.  
  if there aren't any deleted coupons a success email will be sent.

***Db structure***

There are 5 tables 3 main tables and 2 join tables:

* Company (id, comp\_name, password, email) – id is a primary key
* Customer (id, cust\_name, password) – id is a primary key
* Coupon (is, title, start\_date, end\_date, amount, type, message, price, image)  
  id is a primary key
* Customer\_Coupon (cust\_id, coupon\_id) - cust\_id and coupons\_id are primary keys
* Company\_Coupon (comp\_id, coupon\_id) - comp\_id and coupons\_id are primary keys

**Beans:**

There are 3 classes (Company, Customer and Coupon) and one Enum (Coupon type)

***Connection pool***

There are 10 available connections. The connection created once the system is uploaded and are given upon a request. If there aren't any available connection the request process is waiting for a connection.

***DAO layer***

There are 3 DAOs:

* **CompanyDAO interface and CompanyDBDAO that implements it**
  + login(String compName, String password): boolean

Compare the attribute to the DB information. In case the company and the password correct return true

* + createCompany(Company company) : void  
    Insert a record to Company table according to the object information
  + removeCompany(Company company): void  
    Delete the company and all the relevant information from the system.
    - Delete Customer\_Coupon table
    - Delete Coupon table
    - Delete from Company\_Coupon table
    - Delete from Company table
  + updateCompany(Company company): void

Update the DB record according to the information sent as an attribute

* + getCompany(long id): Company  
    Creates a Company object based on the id it gets as an attribute and return it
  + getCompany(String name): Company

Creates a Company object based on the name it gets as an attribute and return it

* + getAllCompanies(): Collection<Company>

Return all companies that exist in the system as a collection

* + getCoupons(Company company): Collection<Coupon>  
    Return all the coupons as a collection that belong to the company sent as an attribute
  + checkDbExistence(Company company): boolean

Return true if a company with the same name exists in the DB

* + isCopuonBelongsCompany(long comp\_id, long coupon\_id): Boolean

The is created to make sure that companies preform actions only on their data.  
Return true id the coupon belongs to the company

* + createCompanyCouponRecord(long comp\_id, long coupon\_id): void

This method is activated when a company creates a coupon and it insert a record to Company\_Coupon table

* + getCouponsByType(Company company, CouponType couponType): Collection<Coupon>

Return a collection of all coupons of Coupon Type

* + getCouponsUpToPrice(Company company, double upToPrice):

Collection<Coupon>  
Return a collection of all coupons with a price less or equals to upToPrice

* + getCouponsUpToDate(Company company, java.util.Date upToDate):

Collection<Coupon>

Return a collection of all coupons with an end date less or equals to upToDate

* **CustomerDAO interface and CustomerDBDAO that implements it**
  + login(String custName, String password): boolean

Compare the attribute to the DB information. In case the customer and the password correct return true

* + createCustomer(Customer customer): void

Insert a record to Customer table according to the object information

* + removeCustomer(Customer customer): void

Delete the customer and all the relevant information from the system.

* + - Delete Customer\_Coupon table
    - Delete from Customer table
  + updateCustomer(Customer customer): void

Update the DB record according to the information sent as an attribute

* + Customer getCustomer(long id): Customer

Creates a Customer object based on the id it gets as an attribute and return it

* + getCustomer(String name): Customer

Creates a Customer object based on the name it gets as an attribute and return it

* + getAllCustomers(): Collection<Customer>

Return all customers that exist in the system as a collection

* + getCoupons(Customer customer): Collection<Coupon>  
    Return all the coupons as a collection that belong to the customer sent as an attribute
  + checkDbExistence(Customer customer): boolean

Return true if a customer with the same name exists in the DB

* + hasAlreadyPurchased(long cust\_id, long coupon\_id): boolean

Return true id the customer already purchased the coupon.  
This method was created in order to prevent the user from purchasing the same coupon more than one time

* + createCustomerCouponRecord(long cust\_id, long coupon\_id): void

This method is activated when a customer purchases a coupon and it insert a record to Customer\_Coupon table

* + getCouponsByType(Customer customer, CouponType couponType): Collection<Coupon>

Return a collection of all coupons of Coupon Type that belong to the customer

* + getCouponsUpToPrice(Customer customer, double upToPrice):

Collection<Coupon>  
Return a collection of all coupons with a price less or equals to upToPrice

* **CouponDAO interface and CouponDBDAO that implements it**
  + createCoupon(Coupon coupon): void

Insert a record to Coupon table according to the object information

* + removeCoupon(Coupon coupon): void

Delete the coupon and all the relevant information from the system.

* + - Delete Customer\_Coupon table
    - Delete from Company\_Coupon table
    - Delete Coupon table
  + updateCoupon(Coupon coupon): void

Update the DB record according to the information sent as an attribute

* + getCoupon(long id): Coupon

Creates a Coupon object based on the id it gets as an attribute and return it

* + getAllCoupons(): Collection<Coupon>

Return all coupons that exist in the system as a collection

* + getAllCouponsByType(CouponType couponType): Collection<Coupon>

Return a collection of all coupons of Coupon Type

* + checkDbExistence(Coupon coupon): Boolean

Return true if a coupon with the same name exists in the DB

* + getCouponsUpToDate(java.util.Date upToDate): Collection<Coupon>

Return a collection of all coupons with an end date less or equals to upToDate

***Facade layer***

Each client (Administrator, Company and Customer) has different interface to interact with the system.  
in this phase the business logic is implemented, and each method activate different methods in the 3 DAOs

* **Admin Facade**
  + createCompany(Company company): void  
    in case a company with the same name exists raises and exception otherwise creates a record
  + removeCompany(Company company): void  
    check if the company exists in the DB in case it does remove it otherwise raise and exception
  + updateCompany(Company company): void  
    retrieve the Db company based on the company it gets as a parameter updates the relevant fields and update the DB.  
    In case the company doesn't exists in the system raise and exception
  + getCompany(long comp\_id): Company

retrieve the DB company based on the is it gets as a parameter in case the company doesn’t exists in the DB raise an exception

* + getAllCompanies(): Collection<Company>

retrieve the companies in the DB in case there aren't any companies raise an exception

* + createCustomer(Customer customer): void  
    in case a customer with the same name exists raise and exception otherwise create a record.
  + removeCustomer(Customer customer): void  
    check if the customer exists in the DB in case it does remove it otherwise raise and exception
  + updateCustomer(Customer customer): void  
    retrieve the Db customer based on the customer it gets as a parameter updates the relevant fields and update the DB.  
    In case the customer doesn't exists in the system raise and exception
  + getCustomer(long cust\_id): Customer

retrieve the DB customer based on id it gets as a parameter in case the customer doesn’t exists in the DB raise an exception

* + getAllCustomers(): Collection<Customer>

retrieve the customers in the DB in case there aren't any customers raise an exception

* **Company Facade**
  + createCoupon(Coupon coupon): void   
    in case a coupon with the same name exists raises and exception otherwise creates a record
  + removeCoupon(Coupon coupon): void

In case the coupon that the company asks to remove doesn't exist or the company tries to remove coupon that doesn't belong to her raise an exception otherwise remove the coupon from the system

* + updateCoupon(Coupon coupon): void  
    In case the coupon that the company asks to update doesn't exist or the company tries to update coupon that doesn't belong to her raise an exception otherwise update the coupon
  + viewCompanyInfromation(): Company  
    Retrieve the company information from the DB and return it as an object
  + getCoupon(long coupon\_id): Coupon  
    In case the coupon doesn't exist in system or doesn't belong to the company raise an exception otherwise return the coupon as an object
  + getAllCoupons(): Collection<Coupon>

retrieve the all coupons (the ones belong to the company) from the DB in case there aren't any coupons raise an exception

* + getCouponByType(CouponType couponType): Collection<Coupon>

Return a collection of all coupons of Coupon Type. in case there aren't any coupons raise an exception

* + getCouponByPrice(double price): Collection<Coupon>

Return a collection of all coupons with a price less or equals to upToPrice. in case there aren't any coupons raise an exception

* + getCouponByDate(java.util.Date date): Collection<Coupon>

Return a collection of all coupons with an end date less or equals to UpToDate.

In case there aren't any coupons raise an exception

* **Customer Façade**
  + purchaseCoupon(Coupon coupon): void

Retrieve the Db coupon based on the object it gets as an attribute in case the coupon doesn't exists raise an exception.  
In addition the following scenarios raise an exception:

1. The customer already purchased the coupon
2. The coupon is out of stock
3. The coupon expired

Otherwise insert a record into Customer\_Coupon table

* + getAllPurchasedCoupons(): Collection<Coupon>  
    Return all the purchased coupons as collection
  + getAllPurchasedCouponsByType(CouponType couponType): Collection<Coupon>  
    Return a collection of all coupons of Coupon Type. in case there aren't any coupons raise an exception
  + getAllPurchasedCouponsByPrice(double price): Collection<Coupon>  
    Return a collection of all coupons with a price less or equals to upToPrice. in case there aren't any coupons raise an exception